Exploratory Testing Session 1 Report

ECSE 429 – Software Validation Term Project (Part A)

1. Session Information

Session ID: Exploratory Session 1
Date: 17/01/25
Duration: 45 minutes
Participants:

- Name: David Zhou
- Student ID: 261135446
- Email: david.zhou3@mail.mcgill.ca

Application Under Test: Thingifier REST API Todo List Manager v1.5.2 - /todos endpoint
Environment: Localhost (`http://localhost:4567`)
Tools Used: Terminal (Ghostty), curl, browser (Chrome)

2. Charter

Identify capabilities and areas of potential instability of the “REST API Todo List Manager”.
Identify documented and undocumented “REST API Todo List Manager” capabilities.
For each capability, create scripts to demonstrate the capability.
Exercise each capability identified with data typical to the intended use of the application.

3. Session Notes

The script, unit test, and video demo for the following session notes can be found on Github https://github.com/Rampex1/ECSE429
- p1/scripts/session1_todos_exploration.sh
- p1/tests/TodosApiTest.java
- p1/tests/ServiceAvailabilityTest.java
- p1/videos/session1_session2_video.mov
or alternatively,
https://drive.google.com/drive/folders/1BTJ8P-kmfvu9HgpwTECzJA8jzKxz3Ekh?usp=drive_link

The following session notes document tests designed in accordance with the session charter to identify documented and undocumented capabilities, explore areas of instability, and exercise the Todo API using typical usage data.

Method	Method Type	is documented?	Expected Behavior
/todos	GET	DOCUMENTED	Return all todo items in the system
/todos	POST	DOCUMENTED	Create new todo, auto-assign an ID
/todos/{id}	GET	DOCUMENTED	Return the specific todo with that ID
/todos/{id}	POST	DOCUMENTED	Update fields of an existing todo
/todos/{id}	PUT	DOCUMENTED	Update a todo, required mandatory fields
/todos/{id}	DELETE	DOCUMENTED	Delete the specified todo
/shutdown	GET	DOCUMENTED	Shut down the API server

expected == observed	Observed Behavior	Side Notes	XML & Screenshots
Yes	GET /todos returned a JSON list of default todos	Confirms in-memory data and preloaded entries	p1/screenshots/session1-1
Yes	POST /todos created a new todo and returned 201 Created	ID auto-generated, Location header set	p1/screenshots/session1-1
Yes	GET /todos/{id} returned the correct todo	Response wrapped in “todos” array	p1/screenshots/session1-1
Yes	POST /todos/{id} updated doneStatus successfully	Allows partial update	p1/screenshots/session1-1
No	PUT /todos/{id} failed when title is missing	PUT requires mandatory fields unlike post	p1/screenshots/session1-2
Yes	DELETE /todos/{id} removed the todo	Follow-up GET returned 404	p1/screenshots/session1-2
Yes	Second DELETE	Proper handling of	p1/screenshots/sess

	returned 404 Not Found	invalid operations	ion1-2
Yes	Missing title returned 400 Bad Request	Validation enforced	p1/screenshots/session1-3
Yes	Empty title rejected with validation error	Validation rule enforced	p1/screenshots/session1-3
Yes	Extra field rejected with 400 error	Strict schema enforcement	p1/screenshots/session1-3
Yes	XML Accept header returned XML formatted response	XML output supported	p1/screenshots/session1-3
Partial	XML POST accepted but returned JSON response	Response defaults to JSON	p1/screenshots/session1-3
Yes	Malformed JSON returned 400 error	Exception message exposed	p1/screenshots/session1-4
Yes	/shutdown terminated server	No confirmation message	p1/screenshots/session1-4

4. Summary of Session Findings

- The system supports full CRUD operations on todos:
  - `GET /todos` returns all todos.
  - `POST /todos` creates new todos and auto-assigns IDs.
  - `GET /todos/{id}` retrieves a specific todo.
  - `POST /todos/{id}` updates existing todos.
  - `DELETE /todos/{id}` removes todos.
- Initial data is preloaded on startup and all data is stored only in memory.
- JSON is the default format; XML is supported using `Accept` and `Content-Type` headers.
- Validation rules are enforced:
  - `title` is mandatory.
  - `title` cannot be empty.
  - Unknown fields are rejected.
- Update behavior differs by method:
  - `POST /todos/{id}` supports partial updates.
  - `PUT /todos/{id}` rejects requests missing mandatory fields.
- Error handling is implemented:

    - Missing or empty fields return `400 Bad Request`.
    - Deleted or unknown resources return `404 Not Found`.
    - Malformed JSON returns `400 Bad Request` with an error message.
- The `/shutdown` endpoint immediately terminates the server.

5. Summary of Concerns

- Inconsistent update semantics between `POST` and `PUT` may confuse API users.
- Internal exception details are exposed in error responses.
- XML submission returns JSON by default, which may be unexpected.
- `/shutdown` provides no confirmation message.

6. Test Ideas

- Verify full behavioral differences between `POST` and `PUT` updates.
- Load and stress testing with large numbers of todos.
- Test long strings, Unicode, and emoji in fields.
- Test malformed XML inputs.
- Test system behavior when the service is offline.
- Test concurrency (simultaneous create/update/delete).
- Verify side effects isolation.
- Validate all return codes for invalid operations.
- Restart-based testing to confirm memory-only persistence.